CS420: Analysis of algorithms

Ewan Davies

Colorado State University

Spring 2026

Splay trees

Self-adjusting binary search trees

- **Splay trees** are special binary search trees with proactive restructuring logic that needs no additional metadata
- Instead, **every** access or update restructures the tree
- The central operation is a **splay** which
  - moves the accessed node x to the root
  - while reducing depth along the whole access path
- Worst-case time for one operation is large, amortized performance is excellent
- The original paper by Sleator and Tarjan is excellent [SlTa85]

The splay step

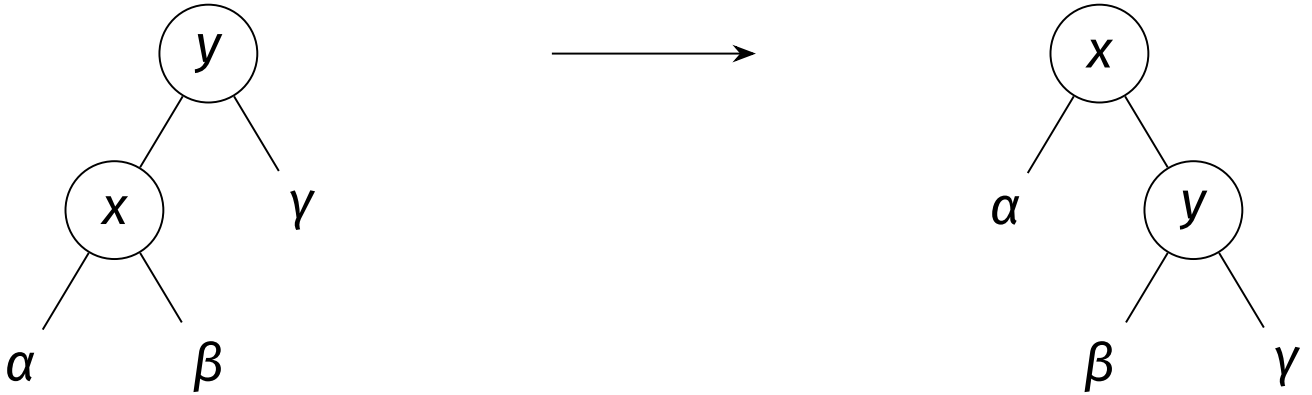
- The **splay** at x repeatedly moves x upward until it becomes the root
- Cases:

  (1) **zig** or **zag**: x has a parent but no grandparent

  (2) **zig-zig** or **zag-zag**: x and its parent are on the same side

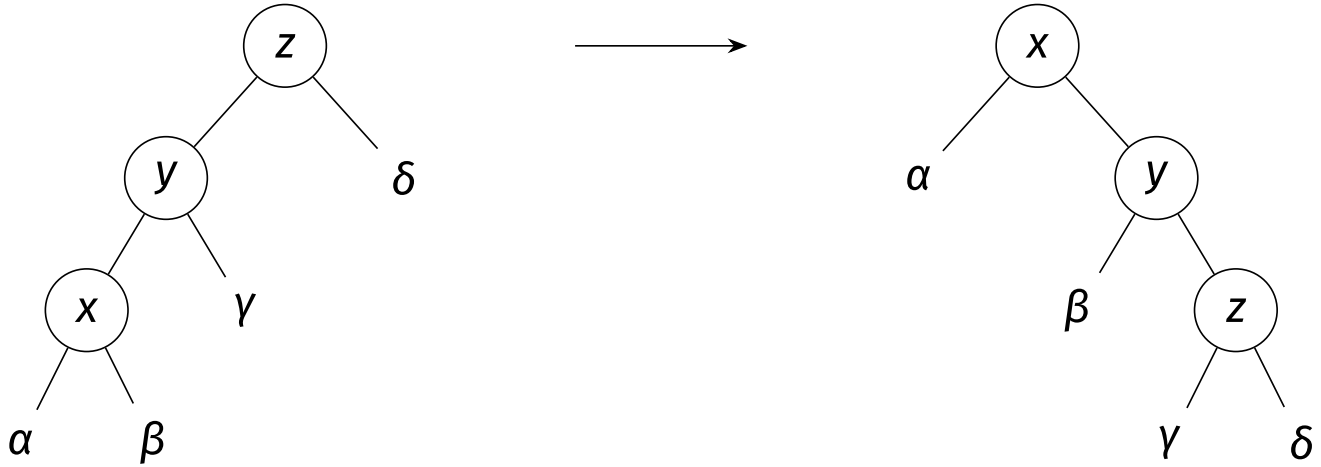
  (3) **zig-zag** or **zag-zig**: x and its parent are on opposite sides
- The effect is not just to move x to the root, but to roughly halve the length of the full access path
- These operations are **carefully designed**: some protocols achieve this and some don't

(1): zig splay step



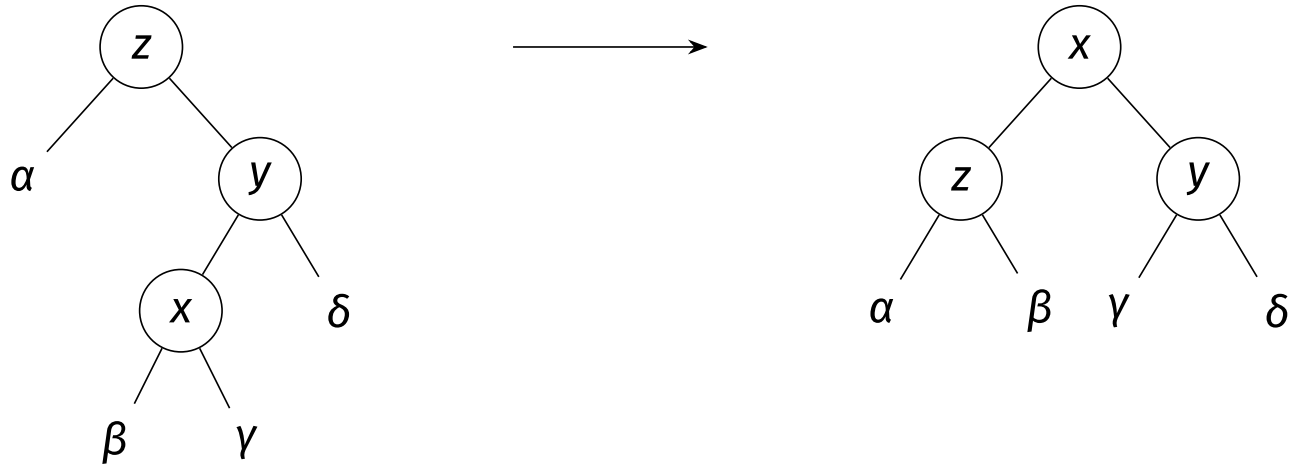
- If x has a parent but no grandparent then we perform a single rotation (**zig**) at its parent to put x at the root
- There is a mirror-image of this case that we can call the **zag**

(2): zig-zig splay step



- If x has a grandparent and both x and its parent are on the same side of their respective parents then a zig at $p^2(x)$ moves x up and another zig at $p(x)$ moves x up again.
- There is a **zag-zag** mirror-image

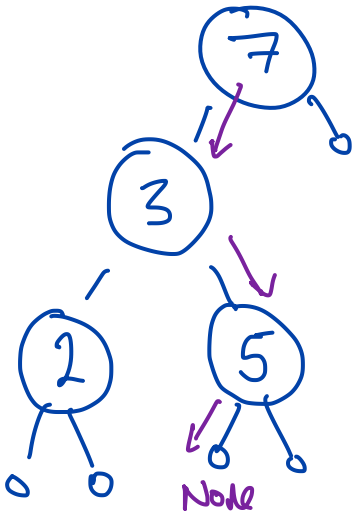
(3): zig-zag step



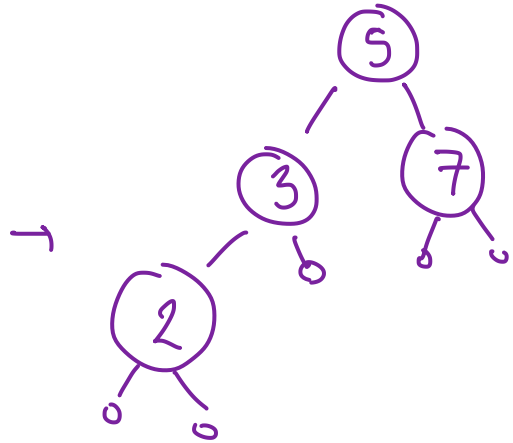
- If x has a grandparent but x and its parent are on opposite sides of their respective parents then first we rotate at $p(x)$ so that x has a new parent, then rotate at this new parent to bring x closer to the root.
- Again, there is a **zag-zig** mirror-image

Exploring examples

access(4)



now splay (5)

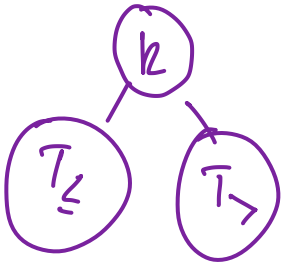


split(k) : $T \longmapsto (T_{\leq}, T_{>})$ where $T_{\leq}$ has keys $\leq k$

$T_{>}$ has keys $> k$

if k not present

insert k: do split(k) return



Splay tree operations

- After a successful lookup or insertion of node x, splay at x
- To delete x, remove it and splay $y = p(x)$
  - Recall that when deleting x we must replace x by its predecessor, which is in turn replaced by its left child
  - After these replacements y is the parent of the predecessor of x, and we splay at y.
- Before splitting at x, first splay at x so that x becomes the root
- The cost of an operation is proportional to the length of the splay path
- We now establish good **amortized** complexity bounds for sequences of operations on splay trees

Accounting and potential methods

- Splay tree analysis uses amortized analysis in a more sophisticated form
- Two equivalent viewpoints:
  - **accounting**: attach credits to nodes
  - **potential function**: track stored structural energy
- If an operation has true cost t and changes the potential from Φ to Φ', then its **amortized cost** is defined to be $t + \Phi' - \Phi$
- This is the true cost t plus the increase in potential $\Phi' - \Phi$ (which is negative if the potential actually decreases)
- Over a sequence, the potential terms cancel in a way we call **telescoping**

Accounting and potential methods

- Consider starting with potential Φ_0 and performing a sequence of m operations with true costs $t_1, \ldots, t_m$
- Let the potential after operation i be Φ_i
- Observe the cancellation if we sum the true costs and the amortized costs

$$\sum_{i=1}^{m} t_i = \Phi_0 - \Phi_m + \sum_{i=1}^{m} \left(t_i + \Phi_i - \Phi_{i-1} \right)$$

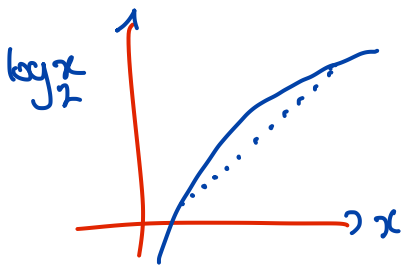
- Then to understand the total cost it suffices to understand amortized costs and the total potential decrease $\Phi_0 - \Phi_m$ over the sequence

A potential function for splay trees

e.g. uniform dist. on n nodes $\Rightarrow w = \frac{1}{n}$ for every node

- Consider a positive real-valued **weight function** w on the internal nodes
- Define **total weights** $t(x)$ as the sum of the individual weights in the subtree rooted at x, including x itself
- We also study a **rank** $r(x) = \log_2 t(x)$, which is *not the same* as red-black tree rank
- The potential function is $\Phi(T) = \sum_{x \in T} r(x)$, where we sum over **internal nodes**

Key math tool: for any $a, b \in \mathbb{R}_{>0}$ $\log_2\left(\frac{a+b}{2}\right) \geq \frac{1}{2}\log_2 a + \frac{1}{2}\log_2 b = \log_2 \sqrt{ab}$



proof:
- $\log_2$ is an increasing function + log rules
- $x^2 \geq 0$ for $x \in \mathbb{R}$

$\Rightarrow \frac{a+b}{2} \geq \sqrt{ab} \iff a+b \geq 2\sqrt{ab}$

$\iff a - 2\sqrt{ab} + b = (\sqrt{a} - \sqrt{b})^2 \geq 0$

Credit accounting

- Given the potential $\Phi(T) = \sum_{x \in T} r(x)$, think of each node x holding on to $r(x)$ "credits"
- We use one credit to pay for each rotation, so keeping track of credits spent will keep track of the big-O time complexity of a sequence of splay tree operations
- Restructuring the tree can change the ranks and may free some credits or require more credits
- Clever choices of w are surprisingly powerful

Access Lemma

w : weight fn on nodes
t : sum of weights in a subtree
r : $\log_2 t$

Lemma: If T has root v and we splay node x, then the splay has amortized cost at most $3(r(v) - r(x)) + 1$.

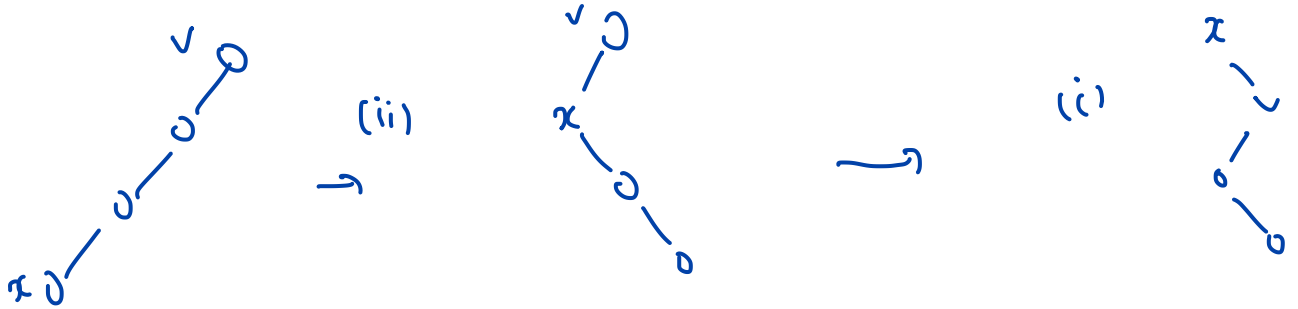
- This is tricky to interpret, but it means the following:
  - Given T, start with $\Phi(T)$ credits
  - Then given a budget of $3(r(v) - r(x)) + 1$ additional credits, we can splay x while paying 1 credit for each rotation
  - At the end we have a restructured tree T' *and* a balance of $\Phi(T')$ credits left
  - It is helpful to think of storing $r(y)$ credits on each internal node y of the tree

Proof I

- Let p be the parent function for a tree T just before a splay step involving the nodes x, $y = p(x)$ and $z = p^2(x)$
- Let the tree after the splay step be called T'
- Let r and t be the rank and total weight functions for T
- Let r' and t' the rank and total weight functions for T'
- There are three cases according to the three types of splay step

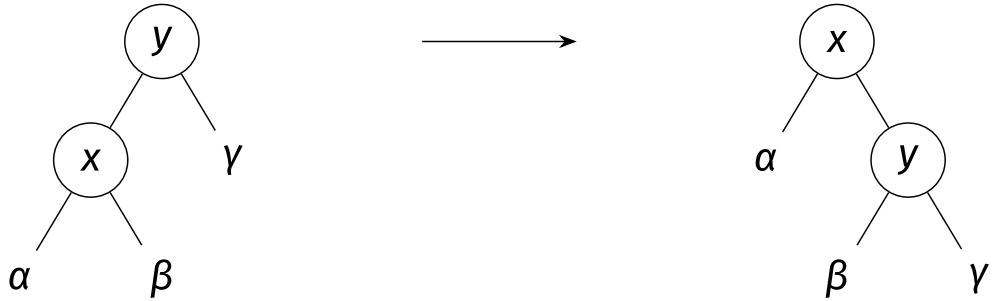
Proof II

- Our desired amortized cost bound is $3(r(v) - r(x)) + 1$ for the entire splay
- We can bound the cost of each type of splay step and sum the costs
- By a telescoping sum, we are done if the **amortized** costs of each splay step satisfy:
  - A type (1) splay step of x costs at most $3(r'(x) - r(x)) + 1$
  - A type (2) or (3) splay step of x costs at most $3(r'(x) - r(x))$



Proof III: splay step (1)

amortized cost time + $\Phi' - \Phi$

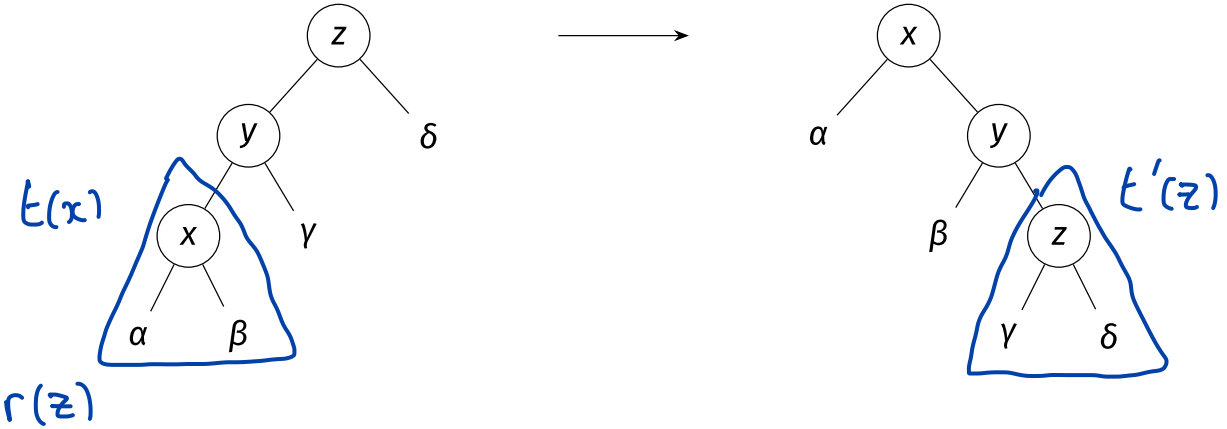


- 1 rotation and only x and y can change rank, so the amortized cost is

$$
\begin{aligned}
& 1 + r'(x) + r'(y) - r(x) - r(y) \\
& \quad \leq 1 + r'(x) - r(x) && \text{since } r(y) \geq r'(y) \\
& \quad \leq 1 + 3(r'(x) - r(x)) && \text{since } r'(x) \geq r(x)
\end{aligned}
$$

- This is within budget

Proof IV: splay step (2)



- Facts: $r'(x) = r(z)$, $r'(x) \geq r'(y)$, $r(y) \geq r(x)$ and $t(x) + t'(z) \leq t'(x)$
- 2 rotations and only x, y, z can change rank, so the amortized cost is

$$2 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z)$$

$$= 2 + r'(y) + r'(z) - r(x) - r(y) \qquad \text{since } r'(x) = r(z)$$

$$\leq 2 + r'(x) + r'(z) - 2r(x). \qquad \text{since } r'(x) \geq r'(y) \text{ and } r(y) \geq r(x)$$

Proof IV: splay step (2)

- This is under budget iff

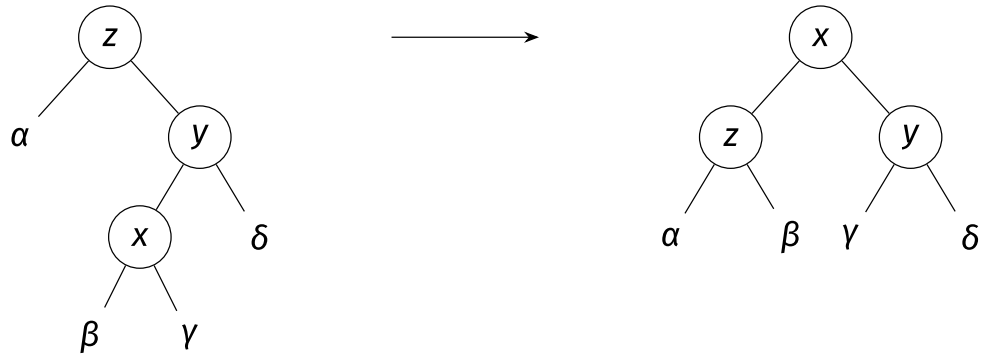
$$2 + r'(x) + r'(z) - 2r(x) \leq 3(r'(x) - r(x))$$

$$\Leftrightarrow r(x) + r'(z) - 2r'(x) \leq -2$$

$$\Leftrightarrow \log_2\left(\frac{t(x)}{t'(x)}\right) + \log_2\left(\frac{t'(z)}{t'(x)}\right) \leq -2$$

- log is concave so $\log(a) + \log(b) \leq 2\log\left(\frac{a+b}{2}\right)$
- Recall that $t(x) + t'(z) \leq t'(x)$
- Then $\log_2\left(\frac{t(x)}{t'(x)}\right) + \log_2\left(\frac{t'(z)}{t'(x)}\right) \leq 2\log(1/2) = -2$ as required

Proof V: splay step (3)



- Facts: $r'(x) = r'(z)$, $r'(x) \geq r'(y)$, $r(y) \geq r(x)$ and $t'(y) + t'(z) \leq t'(x)$
- 2 rotations and only x, y, z can change rank, so the amortized cost is

$$2 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z)$$

$$= 2 + r'(y) + r'(z) - r(x) - r(y) \qquad \text{since } r'(x) = r(z)$$

$$\leq 2 + r'(y) + r'(z) - 2r(x). \qquad \text{since } r'(x) \geq r'(y) \text{ and } r(y) \geq r(x)$$

Proof V: splay step (3)

- This is under budget (in fact under 2/3 of the budget) if

$$2 + r'(y) + r'(z) - 2r(x) \leq 2(r'(x) - r(x))$$

$$\Leftrightarrow r'(y) + r'(z) - 2r'(x) \leq -2$$

$$\Leftrightarrow \log_2\left(\frac{t'(y)}{t'(x)}\right) + \log_2\left(\frac{t'(z)}{t'(x)}\right) \leq -2$$

- log is concave so $\log(a) + \log(b) \leq 2\log\left(\frac{a+b}{2}\right)$
- In this case we have $t'(y) + t'(z) \leq t'(x)$
- Then $\log_2\left(\frac{t'(y)}{t'(x)}\right) + \log_2\left(\frac{t'(z)}{t'(x)}\right) \leq 2\log(1/2) = -2$ as required. □

Consequences of the Access Lemma

Balance Theorem

Theorem: Consider a sequence of m accesses on an n-node splay tree T. The total access time is $O((m + n) \log n + m)$.

Proof:

- Choose $w(x) = 1/n$ for every x and let $W = \sum_{x \in T} w(x) = 1$
- Let T' be the final tree and consider the potential decrease $\Phi(T) - \Phi(T')$
- Each node starts with $r(x) \leq \log_2 W = 0$ and ends with $r'(x) \geq \log_2 w(x) = -\log_2 n$ so $\Phi(T) - \Phi(T') \leq \sum_{x \in T} \log_2 n = n \log_2 n$
- Access Lemma $\Rightarrow$ each amortized cost is at most $3(0 + \log_2 n) + 1 = 3 \log_2 n + 1$.
- Then the total cost is at most $n \log_2 n + 3m \log_2 n + m$. □

Static Optimality Theorem

Theorem: Consider a sequence of m accesses on an n-node splay tree T such that item x is accessed $p_x \geq 1$ times. The total access time is

$$O\left(m + \sum_x p_x \log\left(\frac{m}{p_x}\right)\right).$$

Proof:

- Choose $w(x) = p_x/m$ for every x and let $W = \sum_{x \in T} w(x) = 1$
- Let T' be the final tree and consider the potential decrease $\Phi(T) - \Phi(T')$
- Each node starts with $r(x) \leq \log_2 W = 0$ and ends with $r'(x) \geq \log_2(p_x/m)$ so $\Phi(T) - \Phi(T') \leq \sum_{x \in T} \log_2(m/p_x)$

Static Optimality Theorem

- Access Lemma ⇒ amortized cost of each access of x is at most $3\log_2(m/p_x) + 1$
- Then the total cost is bounded by

$$\sum_x \log_2(m/p_x) + \sum_x p_x(3\log_2(m/p_x) + 1).$$

- This is $O\left(m + \sum_x p_x \log_2(m/p_x)\right)$ since $p_x \geq 1$. □

Static optimality and entropy

$p(x) = \frac{p_x}{m}$

- The entropy of the access distribution is $H = -\sum_x p(x) \log_2 p(x)$
- A *static* binary search tree can be viewed as a prefix code, where the left/right path to a node x is its codeword and its depth is the codeword length
- By the **Shannon source coding theorem**, the expected **length** of any prefix code is lower-bounded by the entropy H
- Therefore, the expected **access cost** in *any* fixed binary search tree is lower-bounded by $\Omega(H)$
- The **Static Optimality Theorem** shows that splay trees achieve a total access time of $O(m(1 + H))$

Static optimality and entropy

- This means the amortized access cost is $O(1 + H)$, perfectly matching the information-theoretic lower bound of the **best possible** static tree up to a constant factor *without needing to know the distribution up front*
- In this sense, splay trees are information-theoretically optimal and adapt to the access sequence's entropy automatically!

Working Set Theorem

indices : $x_1\ x_2\ x_3\ x_4$
accesses: d b a a c a b a b c
d : 0 1 1 0 2 1 2 1 1 2

- For an access sequence of length m, let $d(j)$ be the number of **distinct** items accessed since the last time item x_j was accessed
- If x_j is being accessed for the first time, let $d(j)$ be the number of distinct items accessed so far
- **Theorem:** The total time for m accesses on an n-node splay tree is $O\left(m + n \log n + \sum_{j=1}^{m} \log(1 + d(j))\right)$.
- Intuitively: recently accessed items (the "working set") are kept near the root and are cheap to access again

Proof of the Working Set Theorem

a b a a c a b
w 1 1/4 1/9
access c
w 1/4 1/9 1

- Assign weights $1, 1/4, 1/9, \ldots, 1/n^2$ to the nodes in the order they are *first* accessed
- Upon accessing x_j (currently with weight $1/\ell^2$), reassign its weight to 1
- For each node with weight $1/k^2$ where $k < \ell$, reassign its weight to $1/(k+1)^2$
- This perfectly permutes the weights! The total weight remains $W = \sum_{k=1}^{n} 1/k^2 \leq \pi^2/6 = O(1)$

$\Phi = \sum_x \log_2(t_x)$ max potential drop is $\sum_x \log_2(\frac{W}{w_x})$

- During access j, the weight of item x_j is precisely $w(x_j) = 1/(1+d(j))^2$
- Access Lemma $\Rightarrow$ the amortized cost to access x_j is $O(\log(W/w(x_j))) = O(\log(1+d(j)))$

Scary fact: $\sum_{k=1}^{\infty} \frac{1}{k^2} = \frac{\pi^2}{6}$

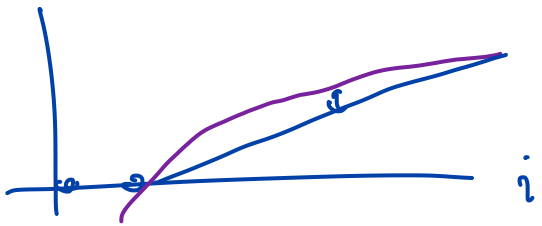
Proof of the Working Set Theorem

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

- The weight reassignment increases the weight of the new root x_j and decreases others, which can only decrease the potential
- The net potential drop over the sequence is at most $\sum_x \log_2(W / w(x)) = O(n \log n)$
- Total cost is bounded by the potential drop plus the sum of amortized costs. □

$$\sum_x \log_2 \frac{W}{w(x)} = \sum_{i=1}^{n} \log_2 (W \cdot i^2) = n \log_2 W + 2 \sum_{i=1}^{n} \log(i)$$

$$\leq O(n) + O(n \log n)$$

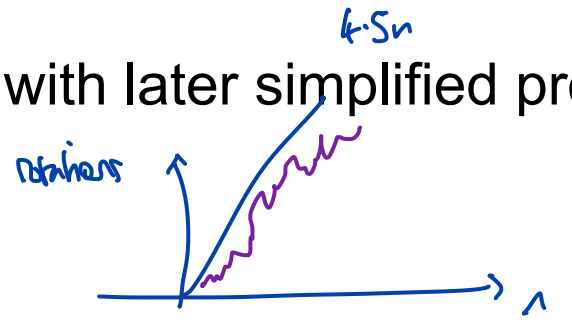


$$\sum_{i=1}^{n} \log(i) = n \cdot \frac{1}{n} \sum_{i=1}^{n} \log(i) \leq n \cdot \log\left(\frac{1}{n} \sum_{i=1}^{n} i\right) = n \log\left(\frac{n+1}{2}\right)$$

↓ Jensen.

Sequential Access Theorem (no proof)

- Also known as the Scanning Theorem
- **Theorem:** Accessing every node of an n-vertex splay tree in sequential (sorted) order takes $O(n)$ time, regardless of the initial shape of the tree.
- This means the amortized cost per access during a sequential scan is remarkably $O(1)$!
- The proof is famously complicated and uses induction and averaging rather than the standard potential function
- It was originally proved by Tarjan [Tarj85], with later simplified proofs by Sundar [Sund92] and Elmasry [Elma04]



$\leq 4.5n$

Dynamic Optimality Conjecture

- Static optimality compares a splay tree to the best *fixed* tree
- What if we compare it to a tree that can dynamically restructure itself, knowing the whole sequence in advance?
- **Conjecture (Sleator and Tarjan, 1985):** The cost of splaying is within a constant factor of the cost of the **optimal** dynamic binary search tree algorithm for any access sequence.
- The Sequential Access Theorem is actually a special case of this conjecture
- This remains one of the most famous open problems in data structures!

Summary of splay tree results

- By choosing different weight functions $w(x)$, we obtain different performance guarantees
- Important examples include:
  - **Balance**: access sequences cost about as much as in a balanced tree
  - **Static Optimality**: near-optimal performance compared to the best static tree for a fixed access distribution
  - **Working Set**: recently accessed items are cheap to access again
  - **Sequential Access**: scanning keys in order takes linear time

Splaysort

- We can sort by:
  - inserting all array items into a splay tree
  - then accessing the keys in sorted order to output them
- In pseudocode:

```
build an empty splay tree T
for x in A:
    insert x into T and splay x
output the keys of T in sorted order
```

- By the Access Lemma and Sequential Access Theorem, the worst-case running time is $O(n \log n)$

References

[Elma04] Elmasry, A. (2004). *On the sequential access theorem and deque conjecture for splay trees*. Theoretical Computer Science, 314 (3), 459–466. DOI: https://doi.org/10.1016/j.tcs.2004.01.019

[SITa85] Sleator, D. D., Tarjan, R. E. (1985). *Self-adjusting binary search trees*. Journal of the ACM, 32 (3), 652–686. DOI: https://doi.org/10.1145/3828.3835

[Sund92] Sundar, R. (1992). *On the deque conjecture for the splay algorithm*. Combinatorica, 12 (1), 95–124. DOI: https://doi.org/10.1007/BF01191208

[Tarj85] Tarjan, R. E. (1985). *Sequential access in splay trees takes linear time*. Combinatorica, 5 (4), 367–378. DOI: https://doi.org/10.1007/BF02579253